Response to reviewers' comments for Paper # 278

We are grateful to all the reviewers for the generous and insightful comments which help us to improve this work to a better level. Hereby, we summarize our responses and corresponding revisions. We wish this feedback could help to clear reviewers' concerns in publishing this paper.

1. RESPONSE TO REVIEWER 1

The main con that I've spotted is the fact that the algorithm might have been more nicely framed also within the Spark environment by taking advantage of the various possibilities offered by it (e.g. caching of RDDs)

Response 1: We agree that utilizing advanced Spark features would improve our solutions. We do not include these optimization in-depth because in this work we set out focus on solving the challenges in parallelizing the GCMP mining algorithms. Nevertheless, we would improve and open-source a Spark-tailored version of SPARE in the future.

2. RESPONSE TO REVIEW 2

The GCMP generalization is not particularly novel. Putting a maximum gap size on consecutive segments is well-known in sequence mining published more than 10 years ago.

Response 2: We agree the fact that the gap parameter is an old concept. Nevertheless, our major contribution in modeling lies in unifying existing movement patterns in the trajectory domain. The gap parameter is naturally introduced to solve the loose-connection anomaly (as in Figure 2). In the revision, we add Section 2.5 to accredit the origin of *gap* concept and distinguish the sequential mining problem with our problem.

In fact, I have doubts about formulating the GCMP patterns as proposed. Are we really interested in all sets of movements beyond a cardinality of size M? Take the Taxi dataset as an example. Let say that there are lots of taxis going from the airport to downtown. Let say that there are 1000 such taxis. For a given M, are we interested in $\binom{1000}{M}$ answers? So this speaks to the problem of picking M. If M is 500, what is $\binom{1000}{M}$? In fact, even if the system gives the single answer of $\binom{1000}{1000}$, I am not sure I am interested in this pattern as I already know that there are many taxis going from the airport to downtown. What I think I am really interested in are GCMP that are "anomalous", which is much harder to define.

Response 3: We agree that if no other parameters are given, the maximum number of patterns discovered could be as large as $\binom{\mathbb{O}}{M}$, which may be less interesting to users. However, such a case rarely occurs due to two reasons. First, in real trajectory mining, parameter M is associated with temporal parameters K, L, G and they jointly prune many false patterns. Second, the nature of our solutions prefer a larger pattern when possible [1]. For example, if (o_1, o_2, o_3) is a pattern and $M = 2$, then both TRPM and SPARE only output (o_1, o_2, o_3) but not any of its subsets. This effectively compresses smaller patterns.

In addition, when the temporal parameters K, L, G are given, a larger M actually decreases the number of resulted patterns. This is because there are less patterns having more than M objects. As shown in Figures 7 (a)(g)(m), a larger M actually reduces the running time of both TRPM and SPARE.

Regarding the second weak point, one line of related work is the superimposition of constraints on spatiotemporal mining. An example is a road network. In other words, given a road network, the network imposes constraints on GCMP.

Response 4: We agree that domain-specific constraints on spatiotemporal mining could be similar to GCMP. However, these domain-specific constraints are nontrivial to be generalized. Since our GCMP model as well as solutions do not leverage domain knowledge, we can support pattern discovery in a broader context. We add the discussion in Section 2.3.

3. RESPONSES TO REVIEWER 3

Though the implementation references Spark as the implementation platform (as it also reflects in github repo provided), the algorithm design is mostly limited to MapReduce, aka only Hadoop, which is a very small subset of Spark. This may have a negative impact on the baseline implementation.

[1]TRPM shrinks an invalid larger pattern until it is valid. SPARE grows a valid smaller pattern until it is about to be invalid. Therefore both algorithms tend to find larger patterns.

Particularly, recent releases of Spark have introduced window functions that can be applied directly in the sliding window scenario here. Certainly, the algorithm has to be redesigned to use DataFrame (and/or Spark SQL) interface, it has been noted that this is a very efficient way to execute window functions in Spark

Response 5: We agree that leveraging more advanced Spark features could further improve the performance of our solutions. We do not heavily leverage these features because we wish to focus on solving how to design parallel GCMP mining algorithms. Nevertheless, we are in progress to develop and open-source a better SPARE version with Spark-tailored optimization.

In the revision, we try to add a DataFrame based TRPM solution which aims to leverage the window functions in Spark-SQL. However, to the best of our efforts, we could not achieve this goal in a short time. The major challenge is that current Spark do not support User Defined Aggregate Function (UDAF) in window functions [2]. Without UDAF, implementing our Line sweep algorithm (Algorithm 1) using Spark primitives (e.g., sum, avg, rank and nth-tile) is nontrivial. Nevertheless, we expect less performance boost from TRPM even with UDAF support. This is because the Line sweep algorithm is not properly reducible, where Spark system would fail to leverage the partial aggregates across multiple windows. Then, it would be equivalent to our current TRPM implementation.

In particular, it would be great to provide the difference in the number of partitions/splits, the amount of processing and memory usage (i.e., vcore and memory seconds) between TRPM and SPARE

Response 6: We add Table 7 to include vcore-seconds and RDD memory usage for both TRPM and SPARE. We also add a clear description on partitions in Section 6.

A plot that breaks down the performance gain by each method would be greatly appreciated by the readers.

Response 7: We complement Figure 8 with the breakdown cost of TRPM. This comparison showcases the benefits of both Star Partition (in mapshufle phase) and Apriori Enumeration (in reduce phase) of SPARE.

Some choices of words may need to be reconsidered: for example, "a bunch of" might not be appropriate in a technical paper.

Response 8: We thank the reviewer for the correction. We have changed many unprofessional terms.

References to star partitioning and apriori pruning are missing. Though these are well known, they need to clearly cited. At least the following reference is missing:

Response 9: We add the corresponding references in Section 5.1 and Section 5.2.

In "In contrast, when utilizing the multicore environment, SPAREP achieves 7 times speedup and SPARES achieves 10 times speedup.", was "multicore" referring to the use of all 16 cores in one of your node? The specification of the machine was not clear.

Response 10: We use all 16 cores in a single node in our settings. We revise the description in Section 6.2.3.

The computation of "eta" was slightly different than that in the paper

Response 11: We have updated the GitHub repository to rectify the typo in the equation.

4. RESPONSE TO META REVIEWER

Novelty: The GCMP generalization is not particularly novel. Please elaborate on the novelty of the work.

Response 12: In addition to the points in Response 2, another novelty of our work lies in the techniques. Although the idea of star-partition and apriori-partition are well-known, linking them together to solve the problem in trajectory domain has not been attempted before. Besides, we input heavy details on these algorithms (e.g., theoretical bounds of partition and anti-monotonicity) which have not been proposed before.

Parameter setting: Are we really interested in all sets of movements beyond a cardinality of size M? Please elaborate on the motivation.

Response 13: Similar responses with Response 3.

Spark implementation: the paper references Spark as the implementation platform but the algorithm is limited to MapReduce. Spark has capabilities beyond MapReduce such as window functions. Please provide an implementation that uses the relevant features of Spark, or a convincing discussion of how these features can be useful and why they were not used.

Response 14: Similar responses with Response 1 and 5.

More details in the performance evaluation: Please provide more details about data partitioning and the effect of skew. Also provide details about how star partitioning and a priori pruning contribute to performance. Please provide references to these two methods.

Response 15: The evaluations are added in Table 7 and Figure 8. Due to the space limit, we only provide the summary of performance skew in Table 9 (i.e., the longest time-taking task vs stand deviation of each task). As per our analysis, the skew is limited because every stars are of similar sizes. The references of apriori and star partition are added accordingly in Section 5.1 and Section 5.2.

[2]JIRA Spakr-8641: Native Spark Window Functions `https://issues.apache.org/jira/browse/SPARK-8641`